1. Implicit wait vs explicit wait – millal kumb?

Implicit wait (nt driver.manage().timeouts().implicitlyWait(...)) seab ühe globaal ootuse kogu sessiooniks — kui element ei leia, siis ootab kuni timeout kuni viskab exception.

Explicit wait (nt WebDriverWait koos ExpectedConditions) ootab konkreetse tingimuse (nähtavus, klikitav jne) täitumist konkreetsel elemendil.

Millal kasutada: kui pead ootama mingi spetsiifilise tingimuse täitumist → explicit; kui soovid lihtsat üldootust elementide leidmiseks, siis implicit. Ent pigem eelistada explicitit, ja vältida mõlema segu.

2. findElement vs findElements – mis juhtub, kui elementi pole?

findElement(...): leiab esimese sobiva elemendi, kui pole, viskab NoSuchElementException.

findElements(...): leiab kõik sobivad elemendid listina; kui pole ühtegi, tagastab tühja listi (ei viska exceptionit).

3. NoSuchElementException vs TimeoutException – põhjused ja vahe.

NoSuchElementException: element ei ole DOM-is või ei leidu, kui findElement proovib.

TimeoutException: ootamine (nt explicit wait) läbi sai, tingimus pole täitunud — element võib olla või mitte, aga ei saavutanud eeldatud seisundit antud ajaga.

Vahe: esimene viitab, et WebDriver kohe ei leia elementi; teine, et ootas määratud ajani, kuid tingimus jäi täitmata.

4. StaleElementReferenceException – põhjused ja tüüplahendused.

Põhjused: element leiti varem, kuid DOM on muutunud (leht reload, AJAX, element asendatud) ning WebElement viide on „aegunud“.

Lahendused: pärast DOM-muutust leia element uuesti, kasuta explicit wait’e, ära hoia pikka aega WebElement viidet kui DOM võib muutuda.

5. CSS vs XPath – miks ~80 % juhtudest CSS eelistatud?

CSS valijad on sageli kiirem- ja stabiilsemad ning loetavamad võrreldes XPath’iga. Stack Overflow +1

XPath peab toetama rohkem traversaalvõimalusi (nt vanem/õde/õdaraam) ning brauserites sageli vähem optimeeritud. club.ministryoftesting.com

Seetõttu kui element on lihtsa ID või klassiga leitav — kasuta CSS; keerukamates olukordades, kus vaja traversaal või teksti põhjal valikut — XPath.

6. Page Object Model – põhiprintsiibid.

Iga lehekülg/tähtis osa rakendusest peab olema oma klass (objekt) testraamistikus, kus asuvad locator’id ja tegevused.

Testid kasutavad neid objektiklasse, mitte otse locator’e testis.

Eesmärk on eraldada testide loogika ja lehe elementide definitsioon, parandada hooldatavust, vähendada duplikaate.

7. Headless Chrome – levinud piirangud & vead.

Mõned brauseri-funktsioonid või liidesed võivad headless-režiimis käituda teisiti (nt visuaalsed animatsioonid, modalid, scrollimine).

Võib esineda raskusi video/flash-sisuga, ning testid võivad eeldada GUI-seisundit, mis puudub.

Vajalik kontrollida, et testid ei sõltu visuaalsetest elementidest, mida headless ei näita identse viisi.

8. Miks implicit + explicit koos teevad üleliigseid viivitusi?

Kui kombineerida global implicit wait + explicit wait, tekib ootamatu ajakulutus: implicit võib põhjustada et iga findElement proovib mitu korda, explicit veel ootab tingimust — kokku võib test ootama jääda rohkem kui vaja.

Soovitatav: kasuta pigem explicit wait’e ja hoia implicit wait võimalikult madalal või ära kasuta.

9. Kuidas jätta Chrome avatuks pärast testi (detach)?

Kasuta brauseri käivitamisel sätted, nt ChromeOptions -/-detach: true (sõltuvalt Selenium versioonist).

Mõnikord pole detach-valik toetatud või töötab erinevalt – võib nõuda käsitsi sulgemist või sessiooni mitte lõpetamist.

10. Java: == vs .equals() Stringide võrdlemisel – millal true?

== võrdleb objektide viiteid (kas viidatakse samale String objektil); .equals() võrdleb sisu (Stringi tegelikud tähemärgid).

== annab true ainult kui mõlemad viited on täpselt samale objektile (nt literal interned string või käsitsi sama objekt viidatud).

.equals() annab true kui Stringide sisu on sama, olenemata sellest, kas objektid on erinevad.

11. Java: List vs ArrayList – miks väljad/argumendid tüübi List?

List on liides (interface), ArrayList üks konkreetne implementeerija.

Kui väljad/argumendid on tüübiga List, siis saab vajadusel muuta implementeerijat (nt LinkedList) ilma API muutmata – parema paindlikkuse ja kapseldusprintsiibiga.

See toetab sõltuvuste vähendamist ja testimist (nt saab kasutada mockimist) – üldisemaks programmeerimispraktikaks on programmeerida liideste kaudu.

12. JUnit: millal assertEquals vs assertTrue(condition)?

assertEquals(expected, actual) kasutad siis kui tead täpset väärtust mida ootad ja tahad võrdsust testida.

assertTrue(condition) kasutad siis kui tahad lihtsalt testida mingit tingimust (näiteks actual > 0).

Küsimus on selgus ja väljendusrikkus: assertEquals annab testi tulemusena selgema veateate kui kahe täpse väärtuse võrdlus ebaõnnestub.

13. WebDriverWait: visibilityOfElementLocated vs presenceOfElementLocated – vahe.

presenceOfElementLocated(By locator) ootab, kuni element on DOM-is olemas, kuid ei pruugi olla nähtav (displayed) või klikitav.

visibilityOfElementLocated(By locator) ootab, kuni element on DOM-is ja nähtav (st nähtav/kuvatud) ning sellega saab tavaliselt suhelda.

Seega kui vajad, et element ilmuks kasutajale nähtavale ja oleks klikitav – kasuta visibilityOf…; kui piisab, et lihtsalt DOM-i sees oleks, mitte tingimata nähtav – kasuta presenceOf….

14. Maven Surefire: kuidas käivitada üks testiklass või -meetod CLI-st?

Näide: mvn -Dtest=MyTestClass test — käivitab ainult MyTestClass.

Üks meetod: mvn -Dtest=MyTestClass#myTestMethod test — käivitab konkreetse meetodi.

15. Lokaatoreid püsivamaks: millal kasutada data- atribuute ja miks?

data-\* atribuudid (nt data-test-id="submitButton") on mõeldud arendajate/testijate jaoks, mitte visuaalse UI tarbeks.

Kui neid kasutada locator’ina, on testid stabiilsemad — välistavad visuaalse stiili/klasside muutumise ja sõltuvad vähem UI muutustest.

Kasuta siis, kui arendustiim toetab ja paigaldab data-\* atribuute elementidele, testidest ei sõltu klassinimedest või stiililistest attribuutidest.